Correction Working with Selected classes from the Java API - 0

A -

```
A
ab
Aa
aba
Abab
```

Let us suppose test string is “aba”.

Lambda expression s.toUpperCase().substring(0,1).equals(“A”); means:
“aba”.toUpperCase().substring(0,1).equals(“A”); =>
“ABA”.substring(0,1).equals(“A”); => “A”.equals(“A”); => true.

Correction Working with Selected classes from the Java API - 1

B - 1947-08-14T23:59:59.999999999

LocalTime.MIN –> {00:00}, LocalTime.MAX –> {23:59:59.999999999}, LocalTime.MIDNIGHT –> {00:00}, LocalTime.NOON –> {12:00}.

date.atTime(LocalTime) method creates a LocalDateTime instance by combining date and time parts.

Correction Working with Selected classes from the Java API - 2

B - [Hello, Hello, Hello]

ArrayList’s 1st and 3rd items are referring to same String instance referred by s [s –> “Hello”] and 2nd item is referring to another instance of String.

String is immutable, which means s.replace(“l”, “L”); creates another String instance “HeLLo” but s still refers to “Hello” [s –> “Hello”].

Correction Working with Selected classes from the Java API - 3

A - P0D

Period.of(0, 0, 0); is equivalent to Period.ZERO. ZERO period is displayed as P0D, other than that, Period components (year, month, day) with 0 values are ignored.

toString()'s result starts with P, and for non-zero year, Y is appended; for non-zero month, M is appended; and for non-zero day, D is appended. P,Y,M and D are in upper case.

Correction Working with Selected classes from the Java API - 4

B - Runtime exception

list.add(0, ‘V’); => char ‘V’ is converted to Character object and stored as the first element in the list. list –> [V].

list.add(‘T’); => char ‘T’ is auto-boxed to Character object and stored at the end of the list. list –> [V,T].

list.add(1, ‘E’); => char ‘E’ is auto-boxed to Character object and inserted at index 1 of the list, this shifts T to the right. list –> [V,E,T].

list.add(3, ‘O’); => char ‘O’ is auto-boxed to Character object and added at index 3 of the list. list –> [V,E,T,O].

list.contains(‘O’) => char ‘O’ is auto-boxed to Character object and as Character class overrides equals(String) method this expression returns true. Control goes inside if-block and executes: list.remove(‘O’);.

remove method is overloaded: remove(int) and remove(Object). char can be easily assigned to int so compiler tags remove(int) method. list.remove(<ASCCI value of ‘O’>); ASCCI value of ‘A’ is 65 (this everybody knows) so ASCII value of ‘O’ will be more than 65.

Correction Working with Selected classes from the Java API - 5

A - CoRe

System.out.println(str1 = str2) has assignment(=) operator and not equality(==) operator.

Correction Working with Selected classes from the Java API - 6

E - Compilation error

Correction Working with Selected classes from the Java API - 7

A - Student[James, 25] Student[James, 27] Student[James, 25] Student[James, 25]

Before you answer this, you must know that there are 5 different Student object created in the memory (4 at the time of adding to the list and 1 at the time of removing from the list). This means these 5 Student objects will be stored at different memory addresses.

remove(Object) method removes the first occurrence of matching object and equals(Object) method decides whether 2 objects are equal or not. equals(Object) method defined in Object class uses == operator to check the equality and in this case as 5 Student objects are stored at different memory location, hence not equal.

Correction Working with Selected classes from the Java API - 8

D - [2, 1]

remove method of List interface is overloaded: remove(int) and remove(Object).

indexOf method accepts argument of Object type, in this case list.indexOf(0) => 0 is auto-boxed to Integer object so no issues with indexOf code. list.indexOf(0) returns 2 (index at which 0 is stored in the list). So list.remove(list.indexOf(0)); is converted to list.remove(2);

remove(int) version is matched, it’s a direct match so compiler doesn’t do auto-boxing in this case. list.remove(2) removes the element at index 2, which is 0.

Correction Working with Selected classes from the Java API - 9

B - 1

break; and continue; are used inside for-loop, hence no compilation error.

Correction Working with Selected classes from the Java API - 10

A - false

toString() method defined in StringBuilder class doesn’t use String literal rather uses the constructor of String class to create the instance of String class.

Correction Working with Selected classes from the Java API - 11

B - An exception is thrown at runtime

ConcurrentModificationException exception may be thrown for following condition:

1. Collection is being iterated using Iterator/ListIterator or by using for-each loop.

And

2. Execution of Iterator.next(), Iterator.remove(), ListIterator.previous(), ListIterator.set(E) & ListIterator.add(E) methods. These methods may throw java.util.ConcurrentModificationException in case Collection had been modified by means other than the iterator itself, such as Collection.add(E) or Collection.remove(Object) or List.remove(int) etc.

For the given code, ‘dryFruits’ list is being iterated using the Iterator.

hasNext() method of Iterator has following implementation:

```java
public boolean hasNext() {
    return cursor != size;
}
```

Where cursor is the index of next element to return and initially it is 0.

1st Iteration: cursor = 0, size = 4, hasNext() returns true. iterator.next() increments the cursor by 1 and returns “Walnut”.

2nd Iteration: cursor = 1, size = 4, hasNext() returns true. iterator.next() increments the cursor by 1 and returns “Apricot”. As “Apricot” starts with “A”, hence dryFruits.remove(dryFruit) removes “Apricot” from the list and hence reducing the list’s size by 1, size becomes 3.

3rd Iteration: cursor = 2, size = 3, hasNext() returns true. iterator.next() method throws java.util.ConcurrentModificationException.

Correction Working with Selected classes from the Java API - 12

B - An exception is thrown at runtime

LocalDate.of(…) method first validates year, then month and finally day of the month.

Correction Working with Selected classes from the Java API - 13

D - [2018-07-11, 2020-04-08]

LocalDate objects can be created by using static method parse and of.

removeIf(Predicate) method was added as a default method in Collection interface in JDK 8 and it removes all the elements of this collection that satisfy the given predicate.

Correction Working with Selected classes from the Java API - 14

E - 11-11-11

date –> {2012-01-11}, period –> {P2M}, date.minus(period) –> {2011-11-11} [subtract 2 months period from {2012-01-11}, month is changed to 11 and year is changed to 2011].

Correction Working with Selected classes from the Java API - 15

A - false:false

equals method declared in Object class has the declaration: public boolean equals(Object). Generally, equals method is used to compare different instances of same class but if you pass any other object, there is no compilation error. Parameter type is Object so it can accept any Java object.

str.equals(sb) => String class overrides equals(Object) method but as “sb” is of StringBuilder type so this returns false.

StringBuilder class doesn’t override equals(Object) method. So Object version is invoked which uses == operator, hence sb.equals(str) returns false as well.

Correction Working with Selected classes from the Java API - 16

C - 0:0

`new StringBuilder(100);` creates a StringBuilder instance, whose internal char array's length is 100 but length() method of StringBuilder object returns the number of characters stored in the internal array and in this case it is 0. So, `sb.length()` returns 0.

sb.toString() is the String representation of StringBuilder instance and in this case as there are no characters inside the StringBuilder object, hence sb.toString() returns an empty String "", so `sb.toString().length()` also returns 0.

Correction Working with Selected classes from the Java API - 17

A - e -> e.getSalary() >= 10000

D - (Employee e) -> { return e.getSalary() >= 10000; }

Jack’s salary is 5000 and Liya’s salary is 8000. If Employee’s salary is >= 10000 then that Employee object is removed from the list.

Allowed lambda expression is:

(Employee e) -> { return e.getSalary() >= 10000; },

Can be simplified to: (e) -> { return e.getSalary() >= 10000; } => type can be removed from left side of the expression.

Further simplified to: e -> { return e.getSalary() >= 10000; } => if there is only one parameter in left part, then round brackets (parenthesis) can be removed.

Further simplified to: e -> e.getSalary() >= 10000 => if there is only one statement in the right side then semicolon inside the body, curly brackets and return statement can be removed. But all 3 [return, {}, ;] must be removed together.

Correction Working with Selected classes from the Java API - 18

A - [A, B, C, D]

list1 –> [A, D],

list2 –> [B, C],

Correction Working with Selected classes from the Java API - 19

D - It will print any int value between 0 and 59

Correction Working with Selected classes from the Java API - 20

C - [Counter-5]

Let's see what is happening during execution:

main(String [] args) method goes on to the top of the STACK.

1. ArrayList original = new ArrayList<>(); => It creates an ArrayList object [suppose at memory location 15EE00] and variable 'original' refers to it.

2. original.add(new Counter(10)); => It creates a Counter object [suppose at memory location 25AF06] and adds it as a first element of the ArrayList. This means element at 0th index of the ArrayList instance refers to Counter object at the memory location 25AF06.

3. ArrayList cloned = (ArrayList) original.clone(); => original.clone() creates a new array list object, [suppose at memory location 45BA12] and then it will copy the contents of the ArrayList object stored at [15EE00]. So, cloned contains memory address of the same Counter object.

In this case, original != cloned, but original.get(0) == cloned.get(0). This means both the array lists are created at different memory location but refer to same Counter object.

4. cloned.get(0).count = 5; => cloned.get(0) returns the Counter object stored at the memory location 25AF06 and .count = 5 means change the value of count variable of the Counter object (stored at memory location 25AF06) to 5.

Correction Working with Selected classes from the Java API - 21

C - false:true

isAfter and isBefore method can be interpreted as:

Does 1st Jan 2018 come after 25th Dec 2018? No, false.

Correction Working with Selected classes from the Java API - 22

E -

```
null
Java
Java
<Some text containing @ symbol>
```

Variable ‘arr’ refers to an Object array of size 4 and null is assigned to all 4 elements of this array.

for-loop starts with i = 1, which means at 1st index String instance is stored, at 2nd index StringBuiler instance is stored and at 3rd index SpecialString instance is stored. null is stored at 0th index.

So, first null will be printed on to the console.

String and StringBuilder classes override toString() method, which prints the text stored in these classes. SpecialString class doesn’t override toString() method and hence when instance of SpecialString is printed on to the console, you get: @.

Therefore output will be:

```
null
Java
Java
<Some text containing @ symbol>
```

Correction Working with Selected classes from the Java API - 23

D - swiftCode.substring(4, 6);

substring(int beginIndex, int endIndex) is used to extract the substring. The substring begins at “beginIndex” and extends till “endIndex - 1”.

Correction Working with Selected classes from the Java API - 24

C - Compilation error

‘append’ method is overloaded in StringBuilder class: append(String), append(StringBuffer) and append(char[]) etc.

Correction Working with Selected classes from the Java API - 25

A - ArrayList

D - ArrayList<>

List is an interface so its instance can’t be created using new keyword. List and List<> will cause compilation failure.

ArrayList implements List interface, so it can be it can be used to replace /*INSERT*/. List list = new ArrayList(); compiles successfully.

Starting with JDK 7, Java allows to not specify type while initializing the ArrayList. Type is inferred from the left side of the statement.

Correction Working with Selected classes from the Java API - 26

A - true

Please note that Strings computed by concatenation at compile time, will be referred by String Pool during execution. Compile time String concatenation happens when both of the operands are compile time constants, such as literal, final variable etc.

For the statement, String s2 = “OCAJP” + ““;, `"OCAJP" + ""` is a constant expression as both the operands”OCAJP” and “” are String literals, which means the expression `"OCAJP" + ""` is computed at compile-time and results in String literal “OCAJP”.

So, during compilation, Java compiler translates the statement

String s2 = “OCAJP” + ““;

to

String s2 = “OCAJP”;

As “OCAJP” is a String literal, hence at runtime it will be referred by String Pool.

When Test class is executed,

s1 refers to “OCAJP” (String Pool object).

s2 also refers to same String pool object “OCAJP”.

s1 and s2 both refer to the same String object and that is why s1 == s2 returns true.

Please note that Strings computed by concatenation at run time (if the resultant expression is not constant expression) are newly created and therefore distinct.

For below code snippet:

```
String s1 = "OCAJP";
String s2 = s1 + "";
System.out.println(s1 == s2);
```

Correction Working with Selected classes from the Java API - 27

C - [X, X, Y, Z]

After all the add statements are executed, list contains: [X, Y, X, Y, Z].

list.remove(new String("Y")); removes the first occurrence of "Y" from the list, which means the 2nd element of the list. After removal list contains: [X, X, Y, Z].

NOTE: String class and all the wrapper classes override equals(Object) method, hence at the time of removal when another instance is passes [new String("Y")], there is no issue in removing the matching item.

Correction Working with Selected classes from the Java API - 28

B - -1000-01-01

The minimum supported LocalDate is: {-999999999-01-01} and maximum supported LocalDate is: {+999999999-12-31}.

If period of -3000 years is added to 1st Jan 2000, then result is 1st Jan -1000.

Correction Working with Selected classes from the Java API - 29

A - Compilation error

Though Predicate is a generic interface but raw type is also allowed. Type of the variable in lambda expression is inferred by the generic type of Predicate interface.

In this case, Predicate pr1 = s -> s.length() < 4; Predicate is considered of Object type so variable “s” is of Object type and Object class doesn’t have length() method. So, s.length() causes compilation error.

Correction Working with Selected classes from the Java API - 30

D - false

toString() method defined in StringBuilder class doesn’t use String literal rather uses the constructor of String class to create the instance of String class.

So both s1 and s2 refer to different String instances even though their contents are same. s1 == s2 returns false.

Correction Working with Selected classes from the Java API - 31

A - James Lucy Bill

process(List, Predicate) method prints all the records passing the Predicate’s test and test is to process the records having age greater than 20.

There are 3 records with age > 20 and these are printed in the insertion order.

NOTE: toString() method just returns the name.

Correction Working with Selected classes from the Java API - 32

B - false true

Please note that Strings computed by concatenation at compile time, will be referred by String Pool during execution. Compile time String concatenation happens when both of the operands are compile time constants, such as literal, final variable etc.

Whereas, Strings computed by concatenation at run time (if the resultant expression is not constant expression) are newly created and therefore distinct.

fName is a constant variable and lName is a non-constant variable.

`fName + lName` is not a constant expression and hence the expression will be computed at run-time and the resultant String object "JamesGosling" will not be referred by String Pool.

As fName is constant variable and "Gosling" is String literal, hence the expression `fName + "Gosling"` is a constant expression, therefore expression is computed at compile-time and results in String literal "JamesGosling".

So, during compilation, Java compiler translates the statement

String name2 = fName + "Gosling";

to

String name2 = "JamesGosling";

As "JamesGosling" is a String literal, hence at runtime it will be referred by String Pool.

So, at runtime name1 and name2 refer to different String object and that is why name1 == name2 returns false.

`"James" + "Gosling"` is also a constant expression and hence Java compiler translates the statement

String name3 = "James" + "Gosling";

to

String name3 = "JamesGosling";

This means at runtime, variable ‘name3’ will refer to the same String pool object “JamesGosling”, which is referred by variable ‘name3’.

So, name2 and name3 refer to same String object and that is why name2 == name3 returns true.

Correction Working with Selected classes from the Java API - 33

B - [Walnut, Date]

If you want to remove the items from ArrayList, while using Iterator or ListIterator, then use Iterator.remove() or ListIterator.remove() method and NOT List.remove() method.

In this case ListIterator.remove() method is used. startsWith(“A”) returns true for “Apricot” and “Almond” so these elements are removed from the list. In the output, [Walnut, Date] is displayed.

Correction Working with Selected classes from the Java API - 34

A - 0

‘delete’ method accepts 2 parameters: delete(int start, int end), where start is inclusive and end is exclusive.

This method throws StringIndexOutOfBoundsException for following scenarios:

A. start is negative

B. start is greater than sb.length()

C. start is greater than end

If end is greater than the length of StringBuilder object, then StringIndexOutOfBoundsException is not thrown and end is set to sb.length().

So, in this case, `sb.delete(0, 100);` is equivalent to `sb.delete(0, sb.length());` and this deletes all the characters from the StringBuilder object.

Hence, System.out.println(sb.length()); prints 0 on to the console.

Correction Working with Selected classes from the Java API - 35

A - [10]

Let’s see what is happening during execution:

main(String [] args) method goes on to the top of the STACK.

1. ArrayList original = new ArrayList<>(); => It creates an ArrayList object [suppose at memory location 15EE00] and variable ‘original’ refers to it.

2. original.add(new Integer(10)); => It creates an Integer object [suppose at memory location 25AF06] and adds it as a first element of the ArrayList. This means element at 0th index of the ArrayList instance refers to Integer object at the memory location 25AF06.

3. ArrayList cloned = (ArrayList) original.clone(); => original.clone() creates a new array list object, [suppose at memory location 45BA12] and then it will copy the contents of the ArrayList object stored at [15EE00]. So, cloned contains memory address of the same Integer object.

In this case, original != cloned, but original.get(0) == cloned.get(0). This means both the array lists are created at different memory location but refer to same Integer object.

4. Integer i1 = cloned.get(0); => cloned.get(0) returns the Integer object stored at the memory location 25AF06 and variable ‘i1’ refers to it.

5. ++i1; => As Integer object is immutable, hence ++i1; creates a new Integer object with value 11 and suppose this newly created Integer object is stored at memory location 38AB00. This means variable ‘i1’ stops referring to Integer object at the memory location 25AF06 and starts referring to Integer object at the memory location 38AB00.

Cloned list stays intact and still refers to Integer object at memory location 25AF06.

6. System.out.println(cloned); => Prints [10] on to the console as cloned contains an element which refers to Integer object containing value 10.

Correction Working with Selected classes from the Java API - 36

C - An exception is thrown at runtime

LocalDate.parse(CharSequence) method accepts String in “9999-99-99” format only. Single digit month and day value are padded with 0 to convert it to 2 digits.

To represent 9th June 2018, format String must be “2018-06-09”.

If correct format string is not passed then an instance of java.time.format.DateTimeParseException is thrown.

Correction Working with Selected classes from the Java API - 37

B - Runtime exception

LocalDate.parse(CharSequence text) method accepts String in “9999-99-99” format only, in which month and day part in the passed object referred by text should be of 2 digits, such as to represent MARCH, use 03 and not 3 & to represent 4th day of the month, use 04 and not 4.

Single digit month and day value are not automatically padded with 0 to convert it to 2 digits.

To represent 9th June 2018, format String must be “2018-06-09”.

If you pass “2018-6-9” or “2018-06-9” or “2018-6-09” (not in correct formats), then an instance of java.time.format.DateTimeParseException will be thrown.

In this question, LocalDate.parse(“2018-7-11”) throws an exception at runtime as JULY is represented as 7, whereas it should be represented as 07.

Correction Working with Selected classes from the Java API - 38

A - String

Method m1 is overloaded to accept 3 different parameters: String, CharSequence and Object.

String implements CharSequence and Object is the super Parent class in Java. There is no conflict among the overloaded methods for the call m1(null) as it is mapped to the class lowest in hierarchy, which is String class. Hence, output will be “String”.

Now if you add one more overloaded method, `static void m1(StringBuilder s) {...}` in the Test class, then `m1(null);` would cause compilation error as it would match to both m1(StringBuilder) and m1(String) methods. So m1(null) in that case would be ambiguous call and would cause compilation error.

For the same reason, System.out.println(null); causes compilation error as println method is overloaded to accept 3 reference types Object, String and char [] along with primitive types.

System.out.println(null); matches to both println(char[]) and println(String), so it is an ambiguous call and hence the compilation error.

Correction Working with Selected classes from the Java API - 39

D - 3

StringBuilder class doesn’t override equals(Object) method and hence days.contains(new StringBuilder(“Sunday”)) returns false.

Code inside if-block is not executed and days.size() returns 3.

Correction Working with Selected classes from the Java API - 40

C - true false

This is bit tricky. Just remember this:

Two instances of following wrapper objects, created through auto-boxing will always be same, if their primitive values are same:

Boolean,

Byte,

Character from 000 to 07f (7f equals to 127),

Short and Integer from -128 to 127.

For 1st statement, list.add(27); => Auto-boxing creates an integer object for 27.

For 2nd statement, list.add(27); => Java compiler finds that there is already an Integer object in the memory with value 27, so it uses the same object.

That is why System.out.println(list.get(0) == list.get(1)); returns true.

new Integer(27) creates a new Object in the memory, so System.out.println(list.get(2) == list.get(3)); returns false.

Correction Working with Selected classes from the Java API - 41

B - P2D

of and ofXXX methods are static methods and not instance methods.

Period.of(2, 1, 0) => returns an instance of Period type.

static methods can be invoked using class_name or using reference variable. In this case ofYears(10) is invoked on period instance but compiler uses Period's instance to resolve the type, which is period. A new Period instance {P10Y} is created, after that another Period instance {P5M} is created and finally Period instance {P2D} is created.

This instance is assigned to period reference variable and hence P2D is printed on to the console.

Correction Working with Selected classes from the Java API - 42

A - Line 9

C - Line 8

Line 8's syntax was added in JDK 5 and it compiles without any warnings.

Line 9's syntax was added in JDK 7, in which type parameter can be ignored from right side of the statement, it is inferred from left side, so Line 9 also compiles without any warning.

Type parameter can't be removed from declaration part, hence Line 7 gives compilation error.

Both Line 5 and Line 6 are mixing Generic type with Raw type and hence warning is given by the compiler.

Correction Working with Selected classes from the Java API - 43

B - false false

str1 refers to single space character and isEmpty() method of String returns true if no characters are there in the String. As str1 contains single space, hence b1 is false.

false is first printed on to the console.

str1.trim(); => creates an empty string “” but str1 still refers to single space string ” “.

b1 = str1.isEmpty(); assigns false to b1 and last System.out.println statement prints false on to the console. So output is:

false false

Correction Working with Selected classes from the Java API - 44

A - p -> p.length() >= 1

B - p -> true

C - p -> p.length() < 10

D - p -> !false

p -> true means test method returns true for the passed String.

p -> !false means test method returns true for the passed String.

p -> p.length() >= 1 means test method returns true if passed String's length is greater than or equal to 1 and this is true for all the array elements.

p -> p.length() < 10 means test method returns true if passed String's length is less than 10 and this is true for all the array elements.

Correction Working with Selected classes from the Java API - 45

D - Runtime exception

While working with dates, programmers get confused with M & m and D & d.

Easy way to remember is that Bigger(Upper case) letters represent something bigger. M represents month & m represents minute, D represents day of the year & d represents day of the month.

LocalDate’s object doesn’t have time component, mm represents minute and not months so at runtime format method throws exception.

Correction Working with Selected classes from the Java API - 46

C - false:false:false

“parse” and “of” methods create new instances, so in this case you get 4 different instance of LocalDate stored at 4 different memory addresses.

Correction Working with Selected classes from the Java API - 47

B - [List, Array]

list.add(0, “Array”); means list –> [Array],

list.add(0, “List”); means insert “List” to 0th index and shift “Array” to right. So after this operation, list –> [List, Array]. In the console, [List, Array] is printed.

Correction Working with Selected classes from the Java API - 48

A - Compilation error

list.remove(Object) method returns boolean result but list.remove(int index) returns the removed item from the list, which in this case is of String type and not Boolean type and hence if(list.remove(2)) causes compilation error.

Correction Working with Selected classes from the Java API - 49

C -

```
Student[James, 27]
Student[James, 25]
Student[James, 25]
```

Before you answer this, you must know that there are 5 different Student object created in the memory (4 at the time of adding to the list and 1 at the time of removing from the list).

This means these 5 Student objects will be stored at different memory addresses.

remove(Object) method removes the first occurrence of matching object and equals(Object) method decides whether 2 objects are equal or not. equals(Object) method has been overridden by the Student class and equates the object based on their name and age.

3 matching Student objects are found in the list and 1st list element is removed from the list. Remaining 3 list elements are printed in the insertion order.

Correction Working with Selected classes from the Java API - 50

C - false

equals(String str) method of String class matches two String objects and it takes character’s case into account while matching.

Alphabet A in upper case and alphabet a in lower case are not equal according to this method.

As String objects referred by s1 and s2 have different cases, hence output is false.

Correction Working with Selected classes from the Java API - 51

A - null:true

It is possible to add null to ArrayList instant.

Initially list has 3 elements: [null, null, null].

remove(int) returns the deleted member of the list. In this case `list.remove(0);` returns null as null was deleted from the 0th index. So, list is left with 2 elements: [null, null].

remove(Object) returns true if deletion was successful otherwise false. In this case `list.remove(null)` removes first null from the list and returns true and list is left with just one element: [null].

Hence, the output is: ‘null:true’.

Correction Working with Selected classes from the Java API - 52

C - false:false

isAfter and isBefore method can be interpreted as:

Does 1st Jan 2018 come after 1st Jan 2018? No, false. Does 1st Jan 2018 come before 1st Jan 2018? No, false.

Correction Working with Selected classes from the Java API - 53

B - 2020-02-29

plusMonths(long) method of LocalDate class returns a copy of this LocalDate with the specified number of months added. Negative argument will subtract the passed month(s), hence date.plusMonths(-6) doesn't cause any compilation error.

This method adds the specified amount to the months field in three steps:

```
Add the input months to the month-of-year field

Check if the resulting date would be invalid

Adjust the day-of-month to the last valid day if necessary
```

For the given code,

2020-08-31 plus -6 months would subtract 6 months from the given date and would result in the invalid date 2020-02-31. Instead of returning an invalid result, the last valid day of the month, 2020-02-29, is returned.

Please note, 2020 is leap year and hence last day of February is 29 and not 28.

Correction Working with Selected classes from the Java API - 54

A - 1985-03-16

minusYears, minusMonths, minusWeeks, minusDays methods accept long parameter so you can pass either positive or negative value.

If positive value is passed, then that specified value is subtracted and if negative value is passed, then that specified value is added. I think you still remember: minus minus is plus.

Similarly plusYears, plusMonths, plusWeeks, plusDays methods work in the same manner.

If positive value is passed, then that specified value is added and if negative value is passed, then that specified value is subtracted.

Correction Working with Selected classes from the Java API - 55

C - 2020-09-06

In LocalDate.of(int, int, int) method, 1st parameter is year, 2nd is month and 3rd is day of the month.

toString() method of LocalDate class prints the LocalDate object in ISO-8601 format: “uuuu-MM-dd”.

Correction Working with Selected classes from the Java API - 56

D - 2006-01-29

joiningDate –> {2006-03-16}.

joiningDate.withDayOfYear(29) returns a new LocalDate object with the day of the Year altered.

A year has 365 days, so 29 means 29th day of the year, which is 29th Jan 2006.

NOTE: There are other with methods, you should know for the exam. withDayOfMonth(int), withMonth(int) and withYear(int).

Correction Working with Selected classes from the Java API - 57

C - -28000-01-01

There are 2 of methods available in LocalDate class: of(int, int, int) and of(int, Month, int). Month can either be passed as int value (1 to 12) or enum constants Month.JANUARY to Month.DECEMBER.

Period.parse(CharSequence) method accepts the String parameter in “PnYnMnD” format, over here P,Y,M and D can be in any case. “p-30000y” means Period of -30000 years.

The minimum supported LocalDate is: {-999999999-01-01} and maximum supported LocalDate is: {+999999999-12-31}. If period of -30000 years is added to 1st Jan 2000, then result is 1st Jan -28000.

Correction Working with Selected classes from the Java API - 58

D - Good

When change(String) method is called, both variable s and str refers to same String object.

Line 9 doesn’t modify the passed object instead creates a new String object “Good_Morning”.

But this newly created object is not referred and hence is a candidate for GC.

When control goes back to calling method main(String[]), str still refers to “Good”.

Line 5 prints “Good” on to the console.

Correction Working with Selected classes from the Java API - 59

C - [apple, orange, mango, banana, grape]

remove(Object) method of List interface removes the first occurrence of the specified element from the list, if it is present. If this list does not contain the element, it is unchanged. remove(Object) method returns true, if removal was successful otherwise false.

Initially list has: [apple, orange, grape, mango, banana, grape]. fruits.remove(“grape”) removes the first occurrence of “grape” and after the successful remove, list has: [apple, orange, mango, banana, grape]. fruits.remove(“grape”) returns true, control goes inside if block and executes fruits.remove(“papaya”);

fruits list doesn’t have “papaya”, so the list remain unchanged. In the console, you get: [apple, orange, mango, banana, grape].

Correction Working with Selected classes from the Java API - 60

B - Runtime exception

LocalDate object doesn’t contain time part but ISO_DATE_TIME looks for time portion and throws exception at runtime.

For the OCA exam, you can check following DateTimeFormatter types: BASIC_ISO_DATE, ISO_DATE, ISO_LOCAL_DATE, ISO_TIME, ISO_LOCAL_TIME, ISO_DATE_TIME, ISO_LOCAL_DATE_TIME.

Correction Working with Selected classes from the Java API - 61

B - Compilation error

String is a final class so it cannot be extended.

Correction Working with Selected classes from the Java API - 62

A - [Walnut, Apricot, Almond, Date]

In this example, code is trying to remove an item from the list while iterating using traditional for loop so one can think that this code would throw java.util.ConcurrentModificationException.

But note, java.util.ConcurrentModificationException will never be thrown for traditional for loop. It is thrown for for-each loop or while using Iterator/ListIterator.

In this case dryFruits.remove(new StringBuilder(“Almond”)); will never remove any items from the list as StringBuilder class doesn’t override the equals(Object) method of Object class.

StringBuilder instances created at “dryFruits.add(new StringBuilder(”Almond”));” and “dryFruits.remove(new StringBuilder(”Almond”));” are at different memory locations and equals(Object) method returns false for these instances.

Correction Working with Selected classes from the Java API - 63

B - [List]

list.add(0, “Array”); means list –> [Array],

list.set(0, “List”); means replace the current element at index 0 with the passed element “List”. So after this operation, list –> [List]. In the console, [List] is printed.

Correction Working with Selected classes from the Java API - 64

C - List

Generic type can only be reference type and not primitive type, hence List is not a valid syntax.

If you use raw type List or List then Line 3 will give compilation error as list.get(0) will return Object type. Object type cannot be converted to primitive type int, so List and List will cause compilation failure of Line 3.

List is the only correct option left.

Correction Working with Selected classes from the Java API - 65

D - 2018-06-06

date –> {2018-06-06}.

date.minusDays(10); => as LocalDate is immutable, hence a new LocalDate object is created {2018-05-27} but no variable refers to it. date still refers to {2018-06-06}.

2018-06-06 is displayed on to the console.

Correction Working with Selected classes from the Java API - 66

D - []

As list can store only wrapper objects and not primitives, hence

for list.add(110); auto-boxing creates an Integer object {110}.

for list.add(new Integer(110)); as new keyword is used so another Integer object {110} is created.

for 3rd add method call, list.add(110); auto-boxing kicks in and as 110 is between -128 to 127, hence Integer object created at 1st statement is referred.

removeIf(Predicate) method was added as a default method in Collection interface in JDK 8 and it removes all the elements of this collection that satisfy the given predicate.

Boolean expression is : i == 110; in this expression i is wrapper object and 110 is int literal so java extracts int value of wrapper object, i and then equates. As all the 3 objects store 110, hence true is returned. All integer objects are removed form the list.

If list.removeIf(i -> i == new Integer(110)); was used, then all three list elements would return false as object references are equated and not contents.

Correction Working with Selected classes from the Java API - 67

D - System.out.printin(date); is executed more than 6 times

date –> {2000-06-25}. date.getDayOfMonth() = 25, 25 >= 20 is true, hence control goes inside while loop and executes System.out.println(date); statement.

date.plusDays(-1); creates a new LocalDate object {2000-06-24} but date reference variable still refers to {2000-06-25}. date.getDayOfMonth() again returns 25, this is an infinite loop.

Correction Working with Selected classes from the Java API - 68

A - Compilation error

Constructor of LocalDate is declared private so cannot be called from outside, hence new LocalDate(2020, 2, 14); causes compilation failure.

Overloaded static methods “of” and “parse” are provided to create the instance of LocalDate.

LocalTime, LocalDateTime, Period also specify private constructors and provide “of” and “parse” methods to create respective instances.

Correction Working with Selected classes from the Java API - 69

B - 13579

In the boolean expression (predicate.test(i)): i is of primitive int type but auto-boxing feature converts it to Integer wrapper type.

test(Integer) method of Predicate returns true if passed number is an odd number, so given loop prints only odd numbers. for loops works for the numbers from 1 to 10.

Correction Working with Selected classes from the Java API - 70

E - [7, 17, 5]

removeIf(Predicate) method was added as a default method in Collection interface in JDK 8 and it removes all the elements of this collection that satisfy the given predicate.

Predicate’s test method returns true for all the Integers divisible by 10.

Correction Working with Selected classes from the Java API - 71

D -

```
Student[James, 25]
Student[James, 27]
Student[James, 25]
Student[James, 25]
```

Before you answer this, you must know that there are 5 different Student object created in the memory (4 at the time of adding to the list and 1 at the time of removing from the list). This means these 5 Student objects will be stored at different memory addresses.

remove(Object) method removes the first occurrence of matching object and equals(Object) method decides whether 2 objects are equal or not. equals(Object) method has NOT been overridden by the Student class. In fact, equals(Student) is overloaded. But overloaded version is not invoked while equating the Student objects.

equals(Object) method defined in Object class is invoked and equals(Object) method defined in Object class uses == operator to check the equality and in this case as all the Student objects are stored at different memory location, hence not equal.

Nothing is removed from the students list, all the 4 Student objects are printed in the insertion order.

Correction Working with Selected classes from the Java API - 72

A - Only one initializer block causes compilation error.

Even though code seems to be checking the knowledge of ArrayList but it actually checks the knowledge of Polymorphism.

List list = new ArrayList<>(); is valid statement and list can accept any object passing instanceof check for Sellable type.

Rabbit implements Sellable hence new Rabbit() can be added to list.

But as Mammal doesn’t implement Sellable hence new Mammal() can’t be added to list.

Other initializer blocks can be verified on similar lines. So there is only one initializer block, which causes compilation error.

Correction Working with Selected classes from the Java API - 73

C - 11 : !

String class has length() method, which returns number of characters in the String. So length() method returns 11.

String class has charAt(int index) method, which returns character at passed index. str.charAt(10) looks for character at index 10. index starts with 0. ! sign is at index 10.

Hence output is: 11 : !

Correction Working with Selected classes from the Java API - 74

B - [true]

list.add(true); => Auto-boxing converts boolean literal true to Boolean instance containing true. Element at index 0 represents true.

Boolean class code uses equalsIgnoreCase method to validate the passed String, so if passed String is “true” (‘t’, ‘r’, ‘u’ and ‘e’ can be in any case), then boolean value stored in Boolean object is true otherwise false.

list.add(new Boolean(“tRue”)); => Element at index 1 represents true.

list.add(new Boolean(“abc”)); => Element at index 2 represents false.

So initially list contains [true, true, false].

As generic list is used, so list.remove(1) removes the Boolean instance (true) stored at index 1 and returns it. So after this operation list contains [true, false].

For the boolean expression of if-block, Java runtime extracts the stored boolean value using booleanValue() method, which returns true. Control goes inside if-block and executes list.remove(1); This removes element at index 1 so after this operation list contains [true] and [true] is printed on to the console.

Correction Working with Selected classes from the Java API - 75

A - Space Station

sb - > “SpaceStation”

sb.delete(5, 6) -> “Spacetation”

sb.insert(5, ” S”) -> “Space Station”

sb.toString() -> Creates a new String object “Space Station”

“Space Station”.toUpperCase() -> Creates another String object “SPACE STATION” but the String object is not referred and used.

Method invocation on sb modifies the same object, so after insert(5, ” S”) method invocation sb refers to “Space Station” and this is printed to the Console.

Correction Working with Selected classes from the Java API - 76

D - An exception is thrown at runtime

There is no element at index 0 so call to add element at index 1, “trafficLight.add(1,”RED”);” throws an instance of java.lang.IndexOutOfBoundsException.

trafficLight.remove(new Integer(2)); matches with trafficLight.remove(Object) and hence no compilation error.

Correction Working with Selected classes from the Java API - 77

D - true false

Please note that Strings computed by concatenation at compile time, will be referred by String Pool during execution. Compile time String concatenation happens when both of the operands are compile time constants, such as literal, final variable etc.

Whereas, Strings computed by concatenation at run time (if the resultant expression is not constant expression) are newly created and therefore distinct.

For the statement, String str1 = i1 + s1;, i1 is a final variable of int type and s1 is a final variable of String type. Hence, `i1 + s1` is a constant expression which is computed at compile-time and results in String literal “1:ONE”.

This means during compilation, Java compiler translates the statement

String str1 = i1 + s1;

to

String str1 = “1:ONE”;

As “1:ONE” is a String literal, hence at runtime it will be referred by String Pool.

On the other hand, for the statement, String str2 = i2 + s1;, `i2 + s1` is not a constant expression because i2 is neither of primitive type nor of String type, hence it is computed at run-time and returns a non-pool String object “1:ONE”.

As, str1 refers to String Pool object “1:ONE”, hence `str1 == "1:ONE"` returns true, whereas str2 refers to non-Pool String object “1:ONE” and hence `str2 == "1:ONE"` returns false.

Correction Working with Selected classes from the Java API - 78

D - 01234567

`new StringBuilder(5);` creates a StringBuilder instance, whose internal char array's length is 5 but the internal char array's length is adjusted when characters are added/removed from the StringBuilder instance. `sb.append("0123456789");` successfully appends "0123456789" to the StringBuilder's instance referred by sb.

delete method accepts 2 parameters: delete(int start, int end), where start is inclusive and end is exclusive.

This method throws StringIndexOutOfBoundsException for following scenarios:

A. start is negative

B. start is greater than sb.length()

C. start is greater than end

If end is greater than the length of StringBuilder object, then StringIndexOutOfBoundsException is not thrown and end is set to sb.length().

So, in this case, `sb.delete(8, 1000);` is equivalent to `sb.delete(8, sb.length());` and this deletes characters at 8th index (8) and 9th index (9). So remaining characters are: "01234567".

StringBuilder class overrides toString() method, which prints the text stored in StringBuilder instance. Hence, `System.out.println(sb);` prints 01234567 on to the console.

Correction Working with Selected classes from the Java API - 79

B - s -> {return true;}

In the lambda expression’s body, if used, all 3 [return, {}, ;] must be used together.

Correction Working with Selected classes from the Java API - 80

B - An exception is thrown at runtime

ArrayList are different than arrays, though behind the scene ArrayList uses Object[] to store its elements.

There are 2 things related to ArrayList, one is capacity and another is actual elements stored in the list, returned by size() method. If you don't pass anything to the ArrayList constructor, then default capacity is 10 but this doesn't mean that an ArrayList instance will be created containing 10 elements and all will be initialized to null.

In fact, size() method will still return 0 for this list. This list still doesn't contain even a single element. You need to use add method or its overloaded counterpart to add items to the list. Even if you want to add null values, you should still invoke some methods, nothing happens automatically.

In this question, new ArrayList<>(4); creates an ArrayList instance which can initially store 4 elements but currently it doesn't store any data.

Another point you should remember for the certification exam: Addition of elements in ArrayList should be continuous. If you are using add(index, Element) method to add items to the list, then index should be continuous, you simply can't skip any index.

In this case, list.add(0, “Array”); adds “Array” to 0th index. so after this operation list –> [Array]. You can now add at 0th index (existing elements will be shifted right) or you can add at index 1 but not at index 2. list.add(2, “List”); throws an instance of java.lang.IndexOutOfBoundsException.

Correction Working with Selected classes from the Java API - 81

B - Good_Morning

When change method is called, both variable s and sb refers to same StringBuilder object.

Line 9 modifies the passed object and appends “_Morning” to it. As a result s now refers to “Good_Morning” and sb also refers to “Good_Morning” so when control goes back to calling method main(String[]) Line 5 prints “Good_Morning” on to the console.

Correction Working with Selected classes from the Java API - 82

B - 3

String s1 = new String(“Java”); -> Creates 2 objects: 1 String Pool and 1 non-pool. s1 refers to non-pool object.

String s2 = “JaVa”; -> Creates 1 String pool object and s2 refers to it.

String s3 = “JaVa”; -> Doesn’t create a new object, s3 refers to same String pool object referred by s2.

String s4 = “Java”; -> Doesn’t create a new object, s4 refers to String Pool object created at Line 3.

String s5 = “Java”; -> Doesn’t create a new object, s5 also refers to String Pool object created at Line 3.

So, at Line 9, 3 String objects are available in the HEAP memory: 2 String pool and 1 non-pool.

Correction Working with Selected classes from the Java API - 83

B -

Lambda expression for Predicate is: s -> s.length() < 4. This means return true if passed string’s length is < 4.

So first three array elements are printed.

Correction Working with Selected classes from the Java API - 84

B - Compilation error

List is super type and ArrayList is sub type, hence List l = new ArrayList(); is valid syntax.

Animal is super type and Dog is sub type, hence Animal a = new Dog(); is valid syntax. Both depicts Polymorphism.

But in generics syntax, Parameterized types are not polymorphic, this means ArrayList is not super type of ArrayList. Remember this point. So below syntaxes are not allowed:

ArrayList list = new ArrayList(); OR List list = new ArrayList();

Correction Working with Selected classes from the Java API - 85

A - None of the other options

Both fName and lName are of reference type. fName refers to “James” and lName refers to “Gosling”.

In System.out.println() statement, we have used assignment operator (=) and not equality operator (==). So result is never boolean.

fName = lName means copy the contents of lName to fName.

As lName is referring to “Gosling” and so after the assignment, fName starts referring to “Gosling” as well.

System.out.println() finally prints the String referred by fName, which is “Gosling”.

This option is is not available, hence correct answer is “None of the other options”

Correction Working with Selected classes from the Java API - 86

A - true : true

Both the methods “public boolean isEqual(ChronoLocalDate)” and “public boolean equals(Object)” return true if date objects are equal otherwise false.

NOTE: LocalDate implements ChronoLocalDate.

Correction Working with Selected classes from the Java API - 87

E - NullPointerException is thrown at runtime

Default values are assigned to all array elements. As Boolean is of reference type, hence arr[0] = null and arr[1] = null. After addition list contains [null, null].

list.remove(0) removes and returns the Boolean object referring to null. If expression can specify Boolean type, so no compilation error over here. At this point list contains [null].

For the boolean expression of if-block, Java runtime tries to extract the stored boolean value using booleanValue() method, and this throws an instance of NullPointerException as booleanValue() method is invoked on null reference.

Correction Working with Selected classes from the Java API - 88

A -

```
System.out.println(LocalDateTime.now());
```

E -

```
System.out.printin(LocalTime.now());
```

new LocalDate(), new LocalTime() and new LocalDateTime() give compilation error as constructor of these classes are declared private.

System.out.println(LocalDate.now()); => Prints current date only.

System.out.println(LocalTime.now()); => Prints current time only.

System.out.println(LocalDateTime.now()); => Prints current date and time both.

Correction Working with Selected classes from the Java API - 89

C - An exception is thrown at runtime

ConcurrentModificationException exception may be thrown for following condition:

1. Collection is being iterated using Iterator/ListIterator or by using for-each loop.

And

2. Execution of Iterator.next(), Iterator.remove(), ListIterator.previous(), ListIterator.set(E) & ListIterator.add(E) methods. These methods may throw java.util.ConcurrentModificationException in case Collection had been modified by means other than the iterator itself, such as Collection.add(E) or Collection.remove(Object) or List.remove(int) etc.

PLEASE NOTE: for-each loop internally implements Iterator and invokes hasNext() and next() methods.

For the given code, ‘dryFruits’ list is being iterated using for-each loop (internally as an Iterator).

hasNext() method of Iterator has following implementation:

```
public boolean hasNext() {
    return cursor != size;
}
```

Where cursor is the index of next element to return and initially it is 0.

1st Iteration: cursor = 0, size = 4, hasNext() returns true. iterator.next() increments the cursor by 1 and returns “Walnut”.

2nd Iteration: cursor = 1, size = 4, hasNext() returns true. iterator.next() increments the cursor by 1 and returns “Apricot”. As “Apricot” starts with “A”, hence dryFruits.remove(dryFruit) removes “Apricot” from the list and hence reducing the list’s size by 1, size becomes 3.

3rd Iteration: cursor = 2, size = 3, hasNext() returns true. iterator.next() method throws java.util.ConcurrentModificationException.

If you want to successfully remove the items from ArrayList, while using Iterator or ListIterator, then use Iterator.remove() or ListIterator.remove() method and NOT List.remove(…) method. Using List.remove(…) method while iterating the list (using the Iterator/ListIterator or for-each) may throw java.util.ConcurrentModificationException.

Correction Working with Selected classes from the Java API - 90

B - boolean test(T t);

Single abstract method declared in Predicate interface is boolean test(T t);

NOTE: If you are confused, then check other questions on Predicate and from there you will know about the method declared in Predicate interface.

Correction Working with Selected classes from the Java API - 91

B - [orange, mango, banana, grape]

remove(Object) method of List interface removes the first occurrence of the specified element from the list, if it is present. If this list does not contain the element, it is unchanged. remove(Object) method returns true, if removal was successful otherwise false.

Initially list has: [apple, orange, grape, mango, banana, grape]. fruits.remove(“grape”) removes the first occurrence of “grape” and after the successful remove, list has: [apple, orange, mango, banana, grape]. fruits.remove(“grape”) returns true, control goes inside if block and executes fruits.remove(“apple”);

fruits list contains “apple”, so after the removal list has: [orange, mango, banana, grape].

Correction Working with Selected classes from the Java API - 92

D - APRIL:4

date.getMonth() returns the month of the year filed, using Month enum, all the enum constant names are in upper case.

date.getMonthValue() returns the value of the month.

NOTE: month value starts with 1 and it is different from java.util.Date API, where month value starts with 0.

Correction Working with Selected classes from the Java API - 93

B - An exception is thrown at runtime

LocalTime.of(int hour, int minute) creates an instance of LocalTime class. Valid value for hour is: 0 to 23 and valid value for minute is 0 to 59.

java.time.DateTimeException is thrown if invalid values are passed as arguments.

NOTE: There are other overloaded of methods available:

LocalTime of(int hour, int minute, int second) and

LocalTime of(int hour, int minute, int second, int nanoOfSecond).

Valid value for second is: 0 to 59 and valid value for nano second is: 0 to 999,999,999.

Correction Working with Selected classes from the Java API - 94

D - false

Please note that Strings computed by concatenation at compile time, will be referred by String Pool during execution. Compile time String concatenation happens when both of the operands are compile time constants, such as literal, final variable etc.

Whereas, Strings computed by concatenation at run time (if the resultant expression is not constant expression) are newly created and therefore distinct.

`java += world;` is same as `java = java + world;` and `java + world` is not a constant expression and hence is calculated at runtime and returns a non pool String object "JavaWorld", which is referred by variable 'java'.

On the other hand, variable 'javaworld' refers to String Pool object "JavaWorld". As both the variables 'java' and 'javaworld' refer to different String objects, hence `java == javaworld` returns false.

Correction Working with Selected classes from the Java API - 95

A - [HelloWorld!, Hello, HelloWorld!]

ArrayList’s 1st and 3rd items are referring to same StringBuilder instance referred by sb [sb –> {Hello}] and 2nd item is referring to another instance of StringBuilder.

sb.append(“World!”); means sb –> {HelloWorld!}, which means 1st and 3rd items of ArrayList now refers to StringBuilder instance containing HelloWorld!

In the output, [HelloWorld!, Hello, HelloWorld!] is printed.

Correction Working with Selected classes from the Java API - 96

F - VET

list.add(0, ‘V’); => char ‘V’ is converted to Character object and stored as the first element in the list. list –> [V].

list.add(‘T’); => char ‘T’ is auto-boxed to Character object and stored at the end of the list. list –> [V,T].

list.add(1, ‘E’); => char ‘E’ is auto-boxed to Character object and inserted at index 1 of the list, this shifts T to the right. list –> [V,E,T].

list.add(3, ‘O’); => char ‘O’ is auto-boxed to Character object and added at index 3 of the list. list –> [V,E,T,O].

list.contains(‘O’) => char ‘O’ is auto-boxed to Character object and as Character class overrides equals(String) method this expression returns true. Control goes inside if-block and executes: list.remove(3);.

list.remove(3); => Removes last element of the list. list –> [V,E,T].

for(char ch : list) => First list item is Character object, which is auto-unboxed and assigned to ch. This means in first iteration ch = ‘V’; And after this it is simple enhanced for loop. Output is VET.

Correction Working with Selected classes from the Java API - 97

A - true : true

ISO_LOCAL_DATE formatter formats the date without the offset, such as “1947-08-15”.

ISO_DATE formatter formats the date with offset (if available), such as “1947-08-15” or “1947-08-15+05:30”, but remember LocalDate object doesn’t contain any offset information.

In this case, all the three date instances are meaningfully equal.

For the OCA exam, you can check following DateTimeFormatter types: BASIC_ISO_DATE, ISO_DATE, ISO_LOCAL_DATE, ISO_TIME, ISO_LOCAL_TIME, ISO_DATE_TIME, ISO_LOCAL_DATE_TIME.

Correction Working with Selected classes from the Java API - 98

B - Compilation Error

Variable “i” used in lambda expression clashes with another local variable “i” and hence causes compilation error.

Correction Working with Selected classes from the Java API - 99

C - BABA

At Line n1:

sb –> {“B”}

append(…) method in StringBuilder class is overloaded to accept various arguments and 2 such arguments are String and CharSequence. It’s return type is StringBuilder and as StringBuilder class implements CharSequence interface, hence ‘sb.append(“A”)’ can easily be passed as and argument to sb.append(…) method. Line n2 compiles successfully.

At Line n2:

sb.append(sb.append(“A”)); //sb –> {“B”}

sb.append({“BA”}); //sb –> {“BA”}

{“BABA”}

Hence, Line n3 prints BABA

Correction Working with Selected classes from the Java API - 100

C - P1000M

Check the toString() method of Period class. ZERO period is displayed as P0D, other than that, Period components (year, month, day) with 0 values are ignored.

toString()'s result starts with P, and for non-zero year, Y is appended; for non-zero month, M is appended; and for non-zero day, D is appended. P,Y,M and D are in upper case.

NOTE: Period.parse(CharSequence) method accepts the String parameter in “PnYnMnD” format, over here P,Y,M and D can be in any case.

Correction Working with Selected classes from the Java API - 101

C - 3 0

ArrayList can have duplicate elements, so after addition, list is: [SUNDAY, SUNDAY, MONDAY]. days.size() returns 3 so 3 is printed on to the console.

days.clear(); removes all the elements from the days list, in fact days list will be empty after successful execution of days.clear();

So 2nd System.out.println statement prints 0 on to the console.

Correction Working with Selected classes from the Java API - 102

B - Code fails to compile

LocalDate.now(); retrieves the current date from the system clock. There is no issue with this statement.

obj is of LocalDate type and getHour() method is not defined in LocalDate class, it is defined in LocalTime and LocalDateTime class. Hence obj.getHour() causes compilation failure.

Correction Working with Selected classes from the Java API - 103

A - java.time.format

DateTimeFormatter is a part of “java.time.format” package, whereas LocalDate, LocalTime, LocalDateTime and Period are defined inside “java.time” package.

Correction Working with Selected classes from the Java API - 104

B - trim()

ltrim(), rtrim() and trimBoth() are not defined in String class.

trim() method is used for removing leading and trailing white spaces.

Correction Working with Selected classes from the Java API - 105

B - Code fails to compile

LocalDate.now(); retrieves the current date from the system clock. There is no issue with this statement.

obj is of LocalDate type and getHour() method is not defined in LocalDate class, it is defined in LocalTime and LocalDateTime class. Hence obj.getHour() causes compilation failure.

Correction Working with Selected classes from the Java API - 106

A - java.time.format

DateTimeFormatter is a part of “java.time.format” package, whereas LocalDate, LocalTime, LocalDateTime and Period are defined inside “java.time” package.

Correction Working with Selected classes from the Java API - 107

B - trim()

ltrim(), rtrim() and trimBoth() are not defined in String class.

trim() method is used for removing leading and trailing white spaces.

Correction Working with Selected classes from the Java API - 108

C - Three options only

substring(int beginIndex, int endIndex) method of String class extracts the substring, which begins at the specified beginIndex and extends to the character at index endIndex - 1.

This method throws IndexOutOfBoundsException if the beginIndex is negative, or endIndex is larger than the length of this String object, or beginIndex is larger than endIndex. e.g.

“freeway”.substring(4, 7) returns “way”

“freeway”.substring(4, 8) throws IndexOutOfBoundsException

substring(int beginIndex) method of String class extracts the substring, which begins with the character at the specified index and extends to the end of this string.

This method throws IndexOutOfBoundsException if beginIndex is negative or larger than the length of this String object. e.g.

“freeway”.substring(4) returns “way”

“freeway”.substring(8) throws IndexOutOfBoundsException

replace(CharSequence target, CharSequence replacement) method of String class returns a new String object after replacing each substring of this string that matches the literal target sequence with the specified literal replacement sequence. e.g.

“Java”.replace(“a”, “A”) –> returns new String object “JAvA”.

Let’s check all the given options:

“REBUS”.substring(2); [begin = 2, end = 4 (end of the string)], returns “BUS” and hence it is a correct option.

“REBUS”.substring(2, 4); [begin = 2, end = 3 (endIndex - 1)], returns “BU” and hence it is incorrect option.

“REBUS”.substring(2, 5); [begin = 2, end = 4 (endIndex - 1)], returns “BUS” and hence it is a correct option.

“REBUS”.replace(“RE”, ““); It replaces”RE” with empty string “” and returns “BUS”, so it is also a correct option.

“REBUS”.substring(2, 6); Length of “REBUS” = 5 and endIndex = 6, which is greater than 5, hence it will thrown IndexOutOfBoundsException at runtime. Incorrect option

“REBUS”.delete(0, 2); Compilation error as delete(…) method is not available in String class, it is part of StringBuilder class. Incorrect option.

So, total 3 options will replace /*INSERT*/ to print BUS on to the console.

Correction Working with Selected classes from the Java API - 109

A - L

`int indexOf(String str)` method of String class returns the index within this string of the first occurrence of the specified substring.
e.g. “Java”.indexOf(“a”) returns 1.

`char charAt(int index)` method of String class returns the char value at the specified index. e.g. “Java”.charAt(2) returns ‘v’.

Let’s check the given expression:

str.charAt(str.indexOf(“A”) + 1)

= “ALASKA”.charAt(“ALASKA”.indexOf(“A”) + 1)

= “ALASKA”.charAt(0 + 1) //“ALASKA”.indexOf(“A”) returns 0.

= “ALASKA”.charAt(1)

= ‘L’

Hence, L is printed on to the console.

Correction Working with Selected classes from the Java API - 110

B - long

E - int

F - short

Compound declarations are allowed in Java for primitive type and reference type.

Range of byte data type is from -128 to 127, hence if byte is used to replace /*INSERT*/, then y = 200 would cause compilation error as 200 is out of range value for byte type. Hence, byte cannot be used to replace /*INSERT*/.

short, int, long, float & double can replace /*INSERT*/ without causing any error. x + y will evaluate to 207 for short, int and long types whereas, x + y will evaluate to 207.0 for float and double types.

String class has overloaded valueOf methods for int, char, long, float, double, boolean, char[] and Object types. valueOf method returns the corresponding String object and length() method returns number of characters in the String object.

So, `String.valueOf(x + y).length()` in case of short, int and long returns 3, on the other hand, in case of float and double it would return 5.

Hence, only 3 options (short, int and long) print expected output on to the console.

Correction Working with Selected classes from the Java API - 111

F - Compilation error

sb –> {“TOMATO”}

sb.reverse() –> {“OTAMOT”}. reverse() method returns a StringBuilder object.

replace method of StringBuilder class accepts 3 arguments: `replace(int start, int end, String str)`. At Line n1, replace(“O”, “A”) method accepts 2 arguments and hence it causes compilation error.

Correction Working with Selected classes from the Java API - 112

C - [AAA, BBB, AAA]

ArrayList instance referred by ‘list’ stores 3 StringBuilder instances.

removeIf(Predicate<? super E> filter) method was added as a default method in Collection interface in JDK 8 and it removes all the elements of this collection that satisfy the given predicate.

StringBuilder class doesn’t override equals(Object) method. So Object version is invoked, which uses == operator, hence `sb.equals(new StringBuilder("AAA"))` would return false as all 4 StringBuilder instances have been created at four different memory locations.

None of the StringBuilder instances are removed from the list.

StringBuilder class overrides toString() method, which returns the containing String and that is why [AAA, BBB, AAA] will be printed on to the console.

Correction Working with Selected classes from the Java API - 113

E - 8

You need to keep in mind an important point related to String Concatenation:

If only one operand expression is of type String, then string conversion is performed on the other operand to produce a string at run time.

If one of the operand is null, it is converted to the string “null”.

If operand is not null, then the conversion is performed as if by an invocation of the toString method of the referenced object with no arguments; but if the result of invoking the toString method is null, then the string “null” is used instead.

Let’s check the expression of Line n1:

text = text + new A(); –> As text is of String type, hence + operator behaves as concatenation operator.

As text is null, so “null” is used in the Expression.

new A() represents the object of A class, so toString() method of A class is invoked, but as toString() method of A class returns null, hence “null” is used in the given expression.

So, given expression is written as:

text = “null” + “null”;

text = “nullnull”;

Hence, Line n2 prints 8 on to the console.

Correction Working with Selected classes from the Java API - 114

D - [P, E, T]

list –> [P, O, T]

sublist method is declared in List interface:

List subList(int fromIndex, int toIndex)

fromIndex is inclusive and toIndex is exclusive

It returns a view of the portion of this list between the specified fromIndex and toIndex. The returned list is backed by this list, so non-structural changes in the returned list are reflected in this list and vice-versa.

If returned list (or view) is structurally modified, then modification are reflected in this list as well but if this list is structurally modified, then the semantics of the list returned by this method become undefined.

If fromIndex == toIndex, then returned list is empty.

If fromIndex < 0 OR toIndex > size of the list OR fromIndex > toIndex, then IndexOutOfBoundsException is thrown.

list.subList(1, 2) –> [O] (fromIndex is inclusive and endIndex is exclusive, so start index is 1 and end index is also 1). subList –> [O].

At Line n2, `subList.set(0, "E");` => sublist –> [E]. This change is also reflected in the backed list, therefore after this statement, list –> [P, E, T]

`System.out.println(list);` prints [P, E, T] on to the console.

Correction Working with Selected classes from the Java API - 115

G - Two definitions

There are 2 rules related to return types of overriding method:

1. If return type of overridden method is of primitive type, then overriding method should use same primitive type.

2. If return type of overridden method is of reference type, then overriding method can use same reference type or its sub-type (also known as covariant return type).

ArrayList is a subtype of List, hence overriding method can use List or ArrayList as return type. Definitions 1 and 2 are valid.

Please note: even though Son is a subtype of Father, List is not subtype of List. Hence definitions 3 and 4 are NOT valid.

On similar lines, even though GrandSon is a subtype of Father, List is not subtype of List. Hence definitions 5 and 6 are also NOT valid.

List is not subtype of List, definition 7 is NOT valid.

ArrayList is not subtype of List, definition 8 is also NOT valid.

Correction Working with Selected classes from the Java API - 116

A - s1.length() == s2.length()

D - s1.equalsignoreCase(s2)

Let’s check all the statements one by one:

s1.equals(s2): equals(String) method of String class matches two String objects and it takes character’s case into account while matching. Alphabet A in upper case and alphabet a in lower case are not equal according to this method. As String objects referred by s1 and s2 have different cases, hence output is false.

s1.equals(s2.toUpper()): Compilation error as there is no toUpper() method available in String class. Correct method name is: toUpperCase().

s2.equals(s1.toLower()): Compilation error as there is no toLower() method available in String class. Correct method name is: toLowerCase().

s1.length() == s2.length(): length() method returns the number of characters in the String object. s1.length() returns 3 and s2.length() also returns 3, hence output is true.

s1.equalsIgnoreCase(s2): Compares s1 and s2, ignoring case consideration and hence returns true.

s1.contentEquals(s2): String class contains two methods: contentEquals(StringBuffer) and contentEquals(CharSequence). Please note that String, StringBuilder and StringBuffer classes implement CharSequence interface, hence contentEquals(CharSequence) method defined in String class can be invoked with the argument of either String or StringBuilder or StringBuffer. In this case, it is invoked with String argument and hence it is comparing the contents of two String objects. This method also takes character’s case into account while matching. As String objects referred by s1 and s2 have different cases, hence output is false.

Correction Working with Selected classes from the Java API - 117

B - false

String class has following two overloaded replace methods:

1. public String replace(char oldChar, char newChar) {}:

Returns a string resulting from replacing all occurrences of oldChar in this string with newChar. If no replacement is done, then source String object is returned. e.g.

“Java”.replace(‘a’, ‘A’) –> returns new String object “JAvA”.

“Java”.replace(‘a’, ‘a’) –> returns the source String object “Java” (no change).

“Java”.replace(‘m’, ‘M’) –> returns the source String object “Java” (no change).

2. public String replace(CharSequence target, CharSequence replacement) {}:

Returns a new String object after replacing each substring of this string that matches the literal target sequence with the specified literal replacement sequence. e.g.

“Java”.replace(“a”, “A”) –> returns new String object “JAvA”.

“Java”.replace(“a”, “a”) –> returns new String object “Java” (it replaces “a” with “a”).

“Java”.replace(“m”, “M”) –> returns the source String object “Java” (no change).

For Line n1, as both oldChar and newChar are same, hence source String (“Java”) is returned by `"Java".replace('J', 'J');` without any change. flag1 stores true.

For Line n2, even though target and replacement are same but as “J” is found in the source String, hence a new String object “Java” is returned by `"Java".replace("J", "J");` after replacing “J” with “J”. flag2 stores false.

flag1 && flag2 evaluates to false.

Correction Working with Selected classes from the Java API - 118

G - 1984-02-29

plusMonths(long) method of LocalDate class returns a copy of this LocalDate with the specified number of months added.

This method adds the specified amount to the months field in three steps:

```
Add the input months to the month-of-year field
```

```
Check if the resulting date would be invalid
```

```
Adjust the day-of-month to the last valid day if necessary
```

For the given code,

1983-06-30 plus 8 months would result in the invalid date 1984-02-30. Instead of returning an invalid result, the last valid day of the month, 1984-02-29, is returned.

Please note, 1984 is leap year and hence last day of February is 29 and not 28.

Correction Working with Selected classes from the Java API - 119

B - Line n1 causes compilation error

list is of Integer type and variable ‘b’ is of byte type.

At Line n1, b is auto-boxed to Byte and not Integer and List can’t store Byte objects, therefore Line n1 causes compilation error.

list.get(0) returns Integer and `list.get(0) * list.get(0)` is evaluated to int, and variable ‘mul’ is of int type only. Therefore, Line n2 compiles successfully.

Correction Working with Selected classes from the Java API - 120

B - [A, E, I, O, U, A, E, I, O]

Starting with JDK 7, Java allows to not specify type while initializing the ArrayList. As variable list is of List type, therefore type of ArrayList is considered as String. Line n1 compiles successfully.

sublist method is declared in List interface:

List subList(int fromIndex, int toIndex)

fromIndex is inclusive and toIndex is exclusive

It returns a view of the portion of this list between the specified fromIndex and toIndex. The returned list is backed by this list, so non-structural changes in the returned list are reflected in this list and vice-versa.

If returned list (or view) is structurally modified, then modification are reflected in this list as well but if this list is structurally modified, then the semantics of the list returned by this method become undefined.

If fromIndex == toIndex, then returned list is empty.

If fromIndex < 0 OR toIndex > size of the list OR fromIndex > toIndex, then IndexOutOfBoundsException is thrown.

At Line n2, list.subList(0, 4) –> [A, E, I, O] (toIndex is Exclusive, therefore start index is 0 and end index is 3].

list.addAll(list.subList(0, 4)); is almost equal to list.addAll(5, [A, E, I, O]); => Inserts A at index 5, E takes index 6, I takes index 7 and O is placed at index 8. list –> [A, E, I, O, U, A, E, I, O]

Last statement inside main(String []) method prints [A, E, I, O, U, A, E, I, O] on to the console.

Correction Working with Selected classes from the Java API - 121

A - ONE ONE ELEVEN

Given statement:

System.out.println(text.concat(text.concat("ELEVEN")).trim()); //'text' refers to "ONE"

System.out.println(text.concat("ONE ELEVEN").trim()); //As String is immutable, hence there is no change in the String object referred by 'text', 'text' still refers to "ONE"

System.out.println(("ONE ONE ELEVEN").trim()); //'text' still refers to "ONE"

System.out.println("ONE ONE ELEVEN"); //trim() method removes the trailing space in this case

ONE ONE ELEVEN is printed on to the console.

Correction Working with Selected classes from the Java API - 122

C - RISE ABOVE

Initially text refers to “RISE”.

Given statement:

text = text + (text = “ABOVE”);

text = “RISE” + (text = “ABOVE”); //Left operand of + operator is evaluated first, text –> “RISE”

text = “RISE” + “ABOVE”; //Right operand of + operator is evaluated next, text –> “ABOVE”

text = “RISE ABOVE”; //text –> “RISE ABOVE”

Hence `System.out.println(text);` prints ‘RISE ABOVE’ on to the console.

Correction Working with Selected classes from the Java API - 123

C - Only one statement causes compilation error

There are no issues with Line n1 and Line n2, both the statements compile successfully.

String class contains contentEquals(CharSequence) method. Please note that String, StringBuilder and StringBuffer classes implement CharSequence interface, hence contentEquals(CharSequence) method defined in String class cab be invoked with the argument of either String or StringBuilder or StringBuffer.

At Line n3, `str.contentEquals(sb)` is invoked with StringBuilder argument and hence it compiles fine. On execution it would compare the contents of String object and the passed StringBuilder object. As both the String object and StringBuilder object contains same content “Game on”, hence on execution, Line n3 will print true.

contentEquals method is not available in StringBuilder class and hence Line n4 causes compilation error.

equals method declared in Object class has the declaration: `public boolean equals(Object)`. Generally, equals method is used to compare different instances of same class but if you pass any other object, there is no compilation error. Parameter type is Object so it can accept any Java object.

`str.equals(sb)` => It compiles fine, String class overrides equals(Object) method but as ‘sb’ is of StringBuilder type so `str.equals(sb)` would return false at runtime.

`sb.equals(str)` => It also compiles fine, StringBuilder class doesn’t override equals(Object) method. So Object version is invoked which uses == operator, hence `sb.equals(str)` would return false as well at runtime.

Correction Working with Selected classes from the Java API - 124

D - [Okinawa, Manila, Batam, Giza]

remove(Object) method of List interface removes the first occurrence of the specified element from the list, if it is present. If this list does not contain the element, it is unchanged. remove(Object) method returns true, if removal was successful otherwise false. Initially list has: [Austin, Okinawa, Giza, Manila, Batam, Giza]. places.remove(“Giza”) removes the first occurrence of “Giza” and after the successful removal, list has: [Austin, Okinawa, Manila, Batam, Giza]. places.remove(“Giza”) returns true, control goes inside if block and executes places.remove(“Austin”); places list contains “Austin”, so after the removal list has: [Okinawa, Manila, Batam, Giza].

Correction Working with Selected classes from the Java API - 125

C - MITS

According to Javadoc, replace(CharSequence target, CharSequence replacement) method of String class returns a new String object after replacing each substring of this string that matches the literal target sequence with the specified literal replacement sequence. The replacement proceeds from the beginning of the string to the end, for example, replacing “aa” with “b” in the string “aaa” will result in “ba” rather than “ab”.

“MISSS”.replace(“SS”, “T”); returns “MITS”.

Correction Working with Selected classes from the Java API - 126

A - p -> true

B - (String p) -> p.length() < 100

D - p -> !!!!true

E - p -> p.length() >= 1

Interface java.util.function.Predicate declares below non-overriding abstract method:

boolean test(T t);

Let's check all the options one by one:

p -> true ✓ Means test method returns true for the passed String. It will print all the elements of the List.

p -> !!!!true ✓ !!!!true => !!!false => !!true => !false => true, means test method returns true for the passed String. It will print all the elements of the List.

p -> !!false ✗ !!false => !true => false, means test method returns false for the passed String. It will not print even a single element of the list.

p -> p.length() >= 1 ✓ Means test method returns true if passed String's length is greater than or equal to 1 and this is true for all the list elements.

p -> p.length() < 7 ✗ Means test method returns true if passed String's length is less than 7 and this is not true for "whether". "whether" will not be displayed in the output.

(String p) -> p.length() < 100 ✓ Means test method returns true if passed String's length is less than 100 and this is true for all the list elements.

String p -> p.length() > 0 ✗ Round brackets or parenthesis are missing around 'String p'. This causes compilation error.

Correction Working with Selected classes from the Java API - 127

A - PATHETIC

String class has following two overloaded replace methods:

1. public String replace(char oldChar, char newChar) {}:

Returns a string resulting from replacing all occurrences of oldChar in this string with newChar. If no replacement is done, then source String object is returned. e.g.

“Java”.replace(‘a’, ‘A’) –> returns new String object “JAvA”.

“Java”.replace(‘a’, ‘a’) –> returns the source String object “Java” (no change).

“Java”.replace(‘m’, ‘M’) –> returns the source String object “Java” (no change).

2. public String replace(CharSequence target, CharSequence replacement) {}:

Returns a new String object after replacing each substring of this string that matches the literal target sequence with the specified literal replacement sequence. e.g.

“Java”.replace(“a”, “A”) –> returns new String object “JAvA”.

“Java”.replace(“a”, “a”) –> returns new String object “Java” (it replaces “a” with “a”).

“Java”.replace(“m”, “M”) –> returns the source String object “Java” (no change).

As String, StringBuilder and StringBuffer all implement CharSequence, hence instances of these classes can be passed to replace method. Line n1 compiles successfully and on execution replaces “N” with “THET”, and hence Line n1 prints PATHETIC on to the console.

Correction Working with Selected classes from the Java API - 128

E - 4th

`int indexOf(String str, int fromIndex)` method of String class returns the index within this string of the first occurrence of the specified substring, starting at the specified index. e.g.

“alaska”.indexOf(“a”, 1) returns 2

“alaska”.indexOf(“a”, 2) returns 2

“alaska”.indexOf(“a”, 3) returns 5

In the given question, ‘arr’ refers to a String array of size 5. Element at index 0 refers to “1st”, element at index 1 refers to “2nd” and so on.

Let’s solve the given expression of Line n1:

arr[place.indexOf(“a”, 3)]

= arr[“faraway”.indexOf(“a”, 3)] //Starts looking for “a” from index 3 of the given String “faraway” and “a” is found at index 3.

= arr[3]

= “4th” //Array element at index 3 refers to “4th”.

Hence, 4th is printed on to the console.